SuperPack:

Dishonest Majority MPC with Constant Online Communication

Daniel Escudero J.P. Morgan AI Research & J.P. Morgan AlgoCRYPT CoE
Vipul Goyal NTT Research
Antigoni Polychroniadou J.P. Morgan AI Research & J.P. Morgan AlgoCRYPT CoE
Yifan Song Tsinghua University
Chenkai Weng Northwestern University

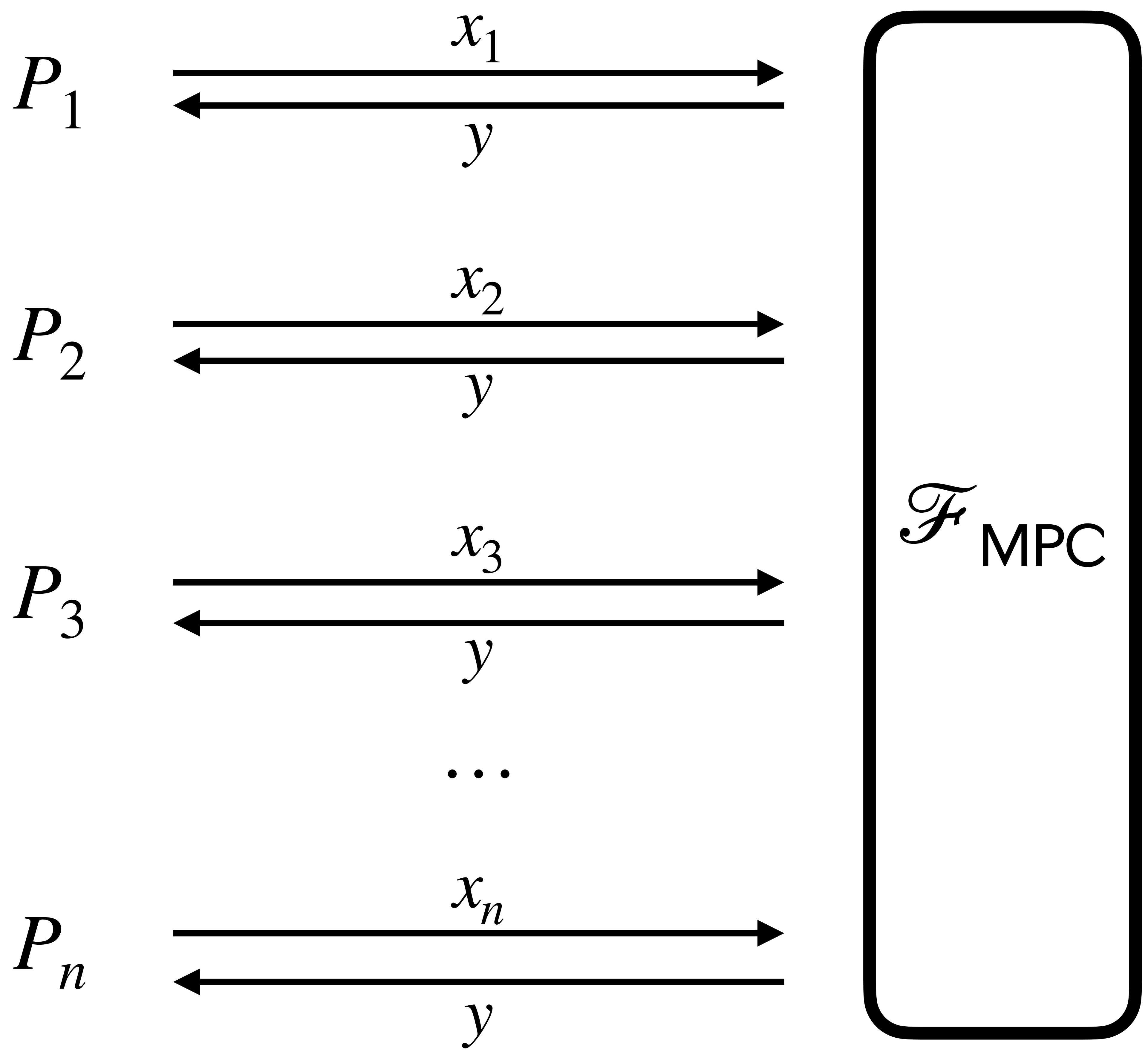
Presented by: Peter Scholl Aarhus University

Secure Multi-Party Computation

A set of n parties $P_1, \ldots, P_n$ securely compute a function

$$y \leftarrow f(x_1, \ldots, x_n)$$

on their private inputs $(x_1, \ldots, x_n)$ while leaking only the output y.



Our MPC Protocol - Setting

- n parties $P_1, \ldots, P_n$
- $y \leftarrow f(x_1, \ldots, x_n)$ is represented by an **arithmetic circuit**
- **Dishonest majority**
  - $t = n(1 - \epsilon)$ **corrupted parties**
  - For constant $\epsilon \in (0,1/2)$
- **Malicious adversary** (secure with abort)

Our Results

[Escudero Goyal Polychroniadou Song Weng 23]

- $O(1)$ online communication per multiplication gate (among all parties)
- Any constant fraction of corruptions (for $0 < \epsilon < 1/2$)
- Communication decreases as the number of honest parties ϵn increases

Online	Circuit-dependent Preprocessing	Circuit-independent Preprocessing
$6/\epsilon$	$4/\epsilon$	$6n + 35/\epsilon$

Communication overhead (number of field elements) per multiplication gate among all parties.

Previous Work

- BeDOZa, SPDZ: all-but-one corruptions
    - Hard to benefit with $\epsilon n > 1$ honest parties. Best: remove $n - t - 1$ parties?
- GPS22: $58/\epsilon + 96/\epsilon^2$ total communication per multiplication gate
    - Benefits from increased ϵn but with large constants
- TinyKeys: MPC for Boolean circuits
    - $n(1 - \epsilon)$ corruptions; still $O(n)$ communication
- TurboPack: $O(1)$ online communication; honest-majority

[BDOZ11] Semi-homomorphic encryption and multiparty computation. Bundling et al. Eurocrypt 2011.
[DPSZ12] Multiparty computation from somewhat homomorphic encryption. Damgård et al. CRYPTO 2012.
[GPS22] Sharing transformation and dishonest majority MPC with packed secret sharing. Goyal et al. CRYPTO 2022.
[HOSS18a] Concretely efficient large-scale MPC with active security (or, TinyKeys for TinyOT). Hazay et al. Asiacrypt 2018.
[HOSS18b] TinyKeys: A new approach to efficient multi-party computation. Hazay et al. CRYPTO 2018.
[EGPS22] TURBOPACK: Honest Majority MPC with Constant Online Communication. Escudero et al. CCS 2022.

Compare with Turbospeedz

n: number of parties. ϵ: percentage of honest parties

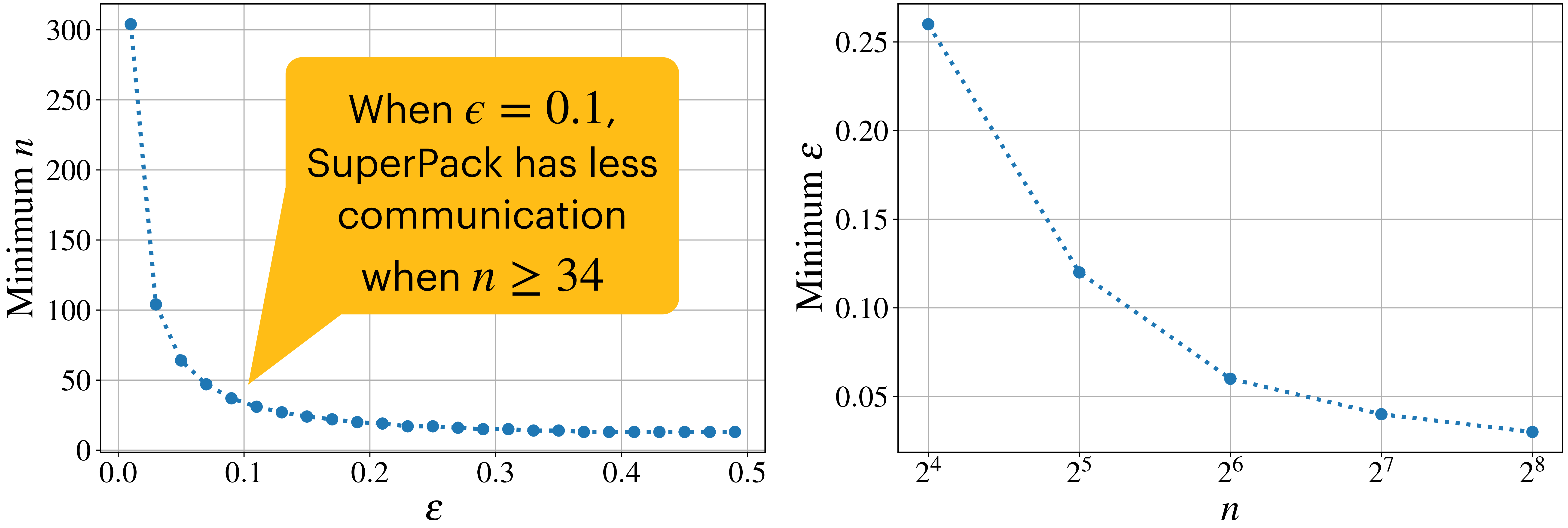
	Online	Circuit-dependent Preprocessing	Circuit-independent Preprocessing
SuperPack	$6/\epsilon$	$4/\epsilon$	$6\mathbf{n} + 35/\epsilon$
Turbospeedz	$2(1-\epsilon)\mathbf{n}$	$4(1-\epsilon)\mathbf{n}$	$6(1-\epsilon)\mathbf{n}$

Communication overhead (number of field elements) per multiplication gate among all parties.
We assume that the preprocessing phase of Turbospeedz is instantiated by Le Mans.
The cost of VOLE/OLE is ignored.

Turbospeedz: Double your online SPDZ! Improving SPDZ using function dependent preprocessing. Ben-Efraim et al. ACNS 2019.
Le mans: Dynamic and fluid MPC for dishonest majority. Rachuri et al. CRYPTO 2022.

Compare with Turbospeedz - Online

Requirements on parameters (n, ϵ) in order to outperform Turbospeedz*



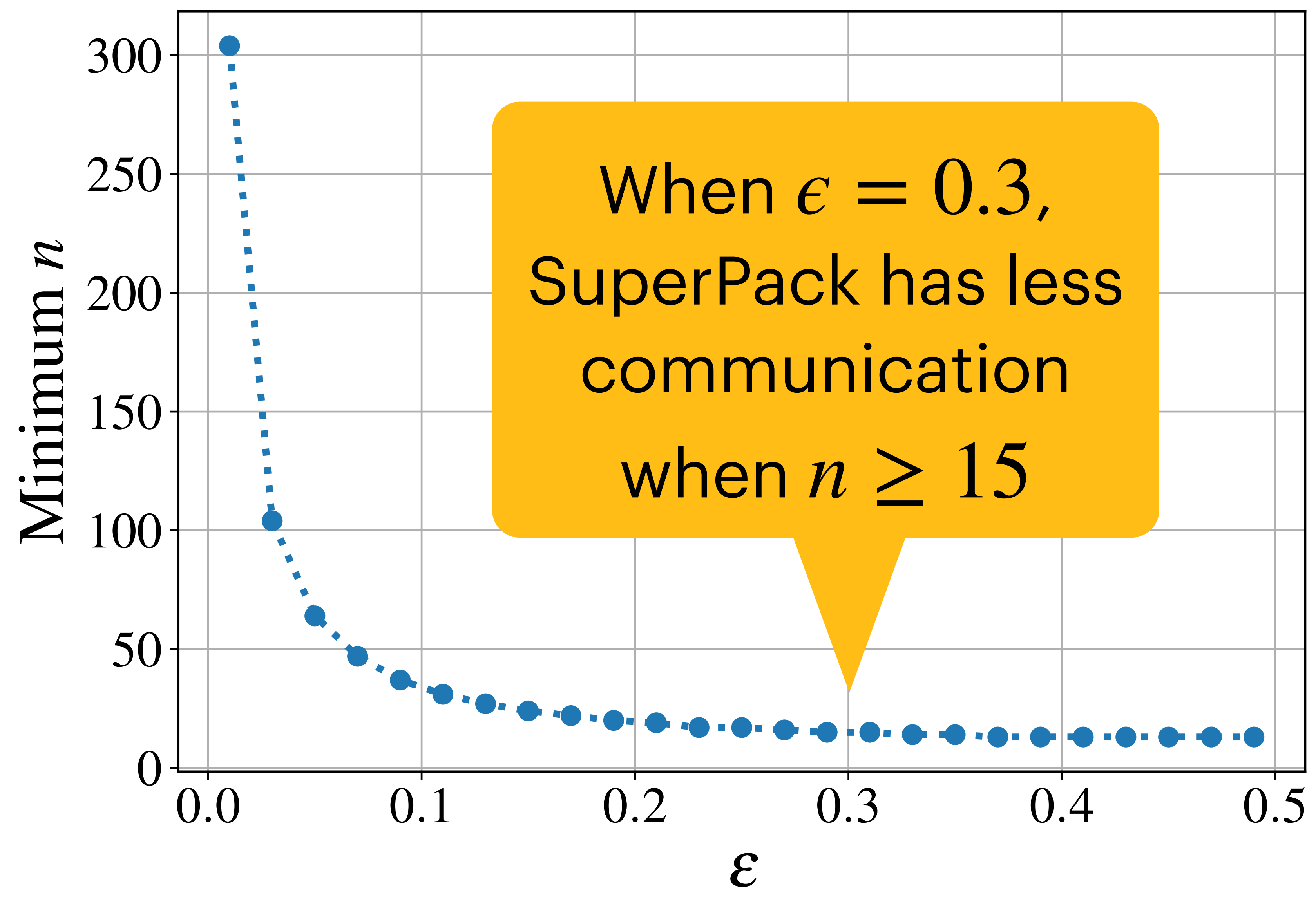
The larger ϵ,
the more honest parties,
the less n needed to outperform Turbospeedz.

The larger n,
the less honest parties
needed to outperform Turbospeedz.

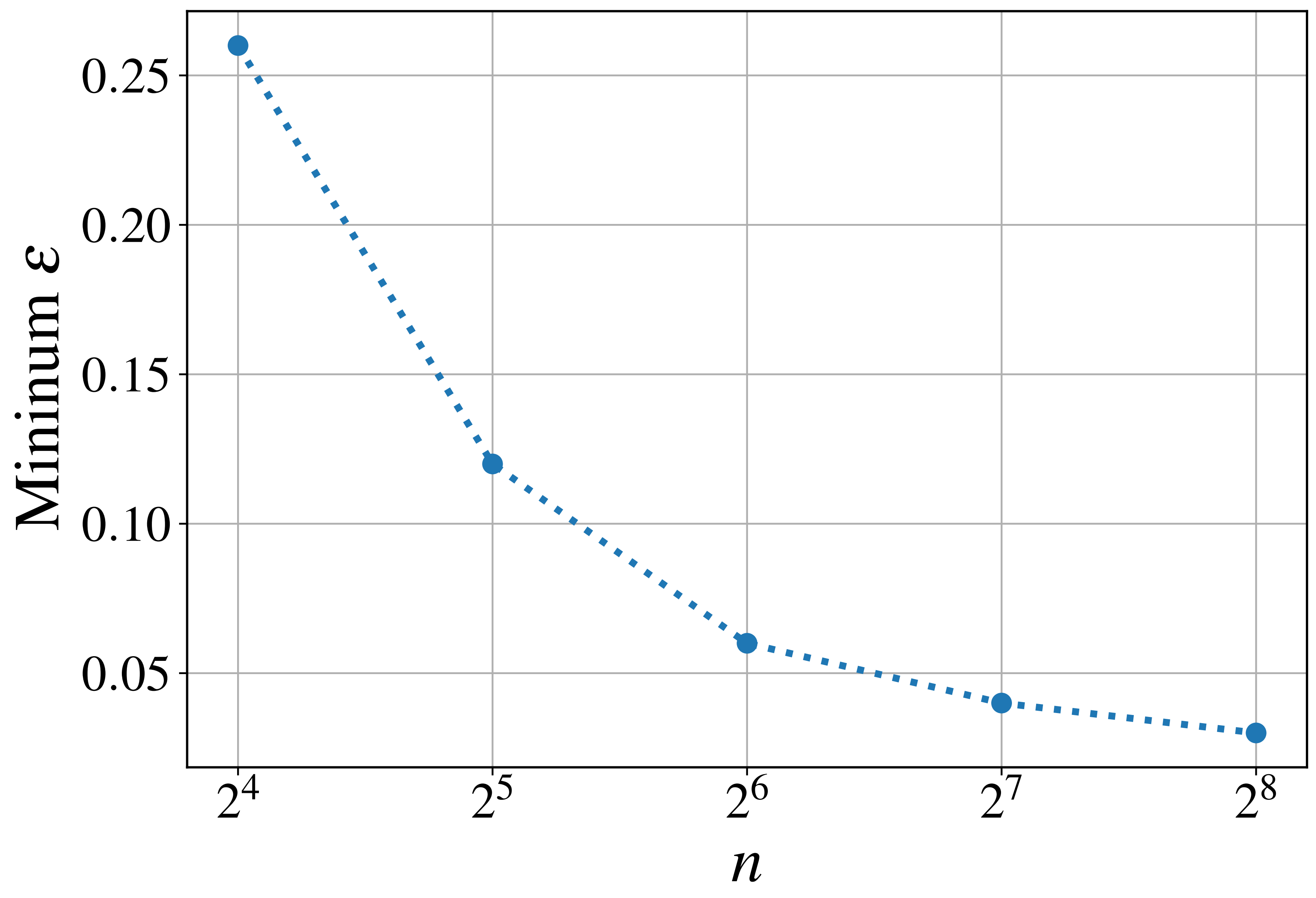
Turbospeedz: Double your online SPDZ! Improving SPDZ using function dependent preprocessing. Ben-Efraim et al. ACNS 2019.
Le mans: Dynamic and fluid MPC for dishonest majority. Rachuri et al. CRYPTO 2022.

Compare with Turbospeedz - Online

Requirements on parameters (n, ϵ) in order to outperform Turbospeedz*



The larger ϵ,
the more honest parties,
the less n needed to outperform Turbospeedz.

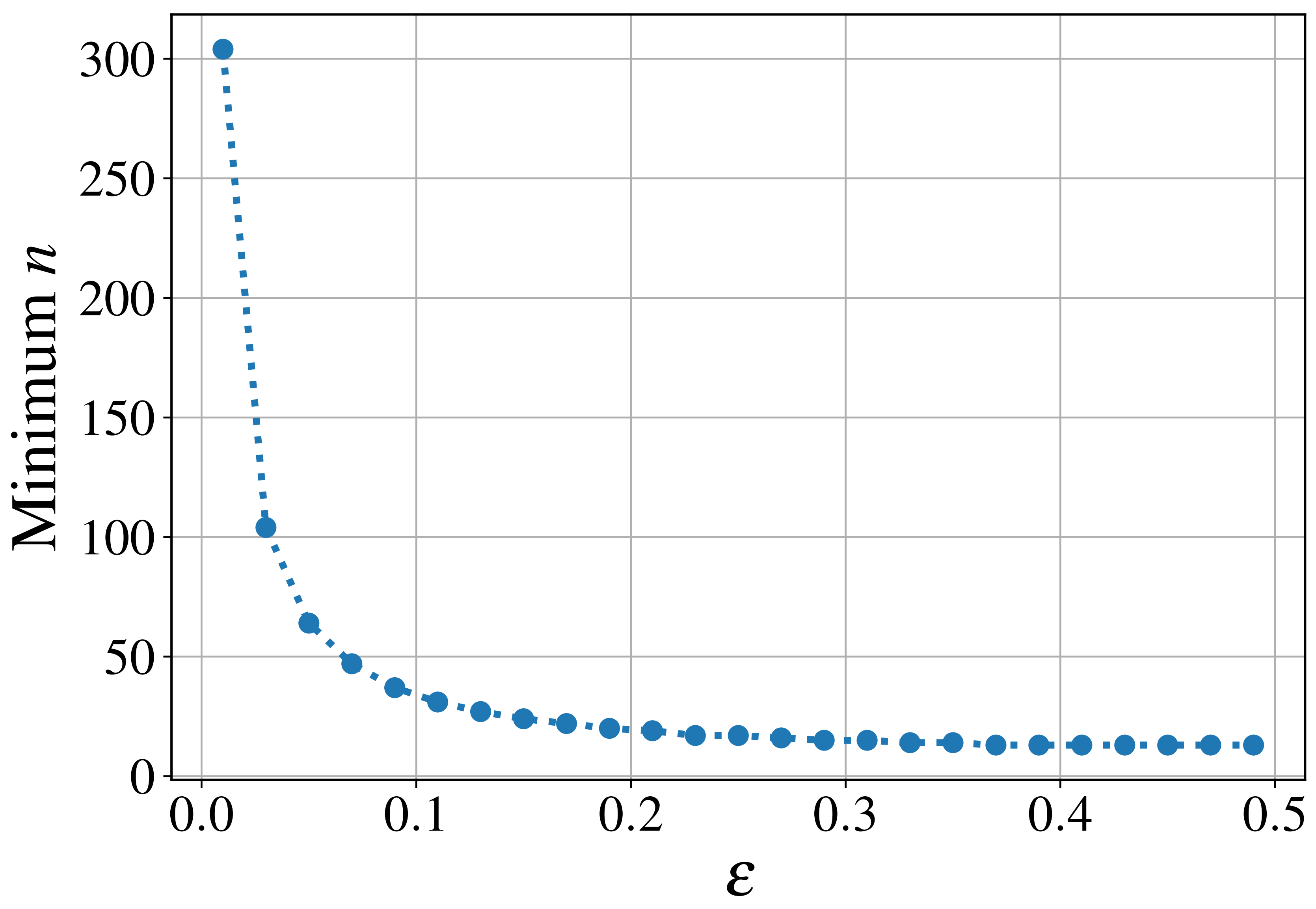


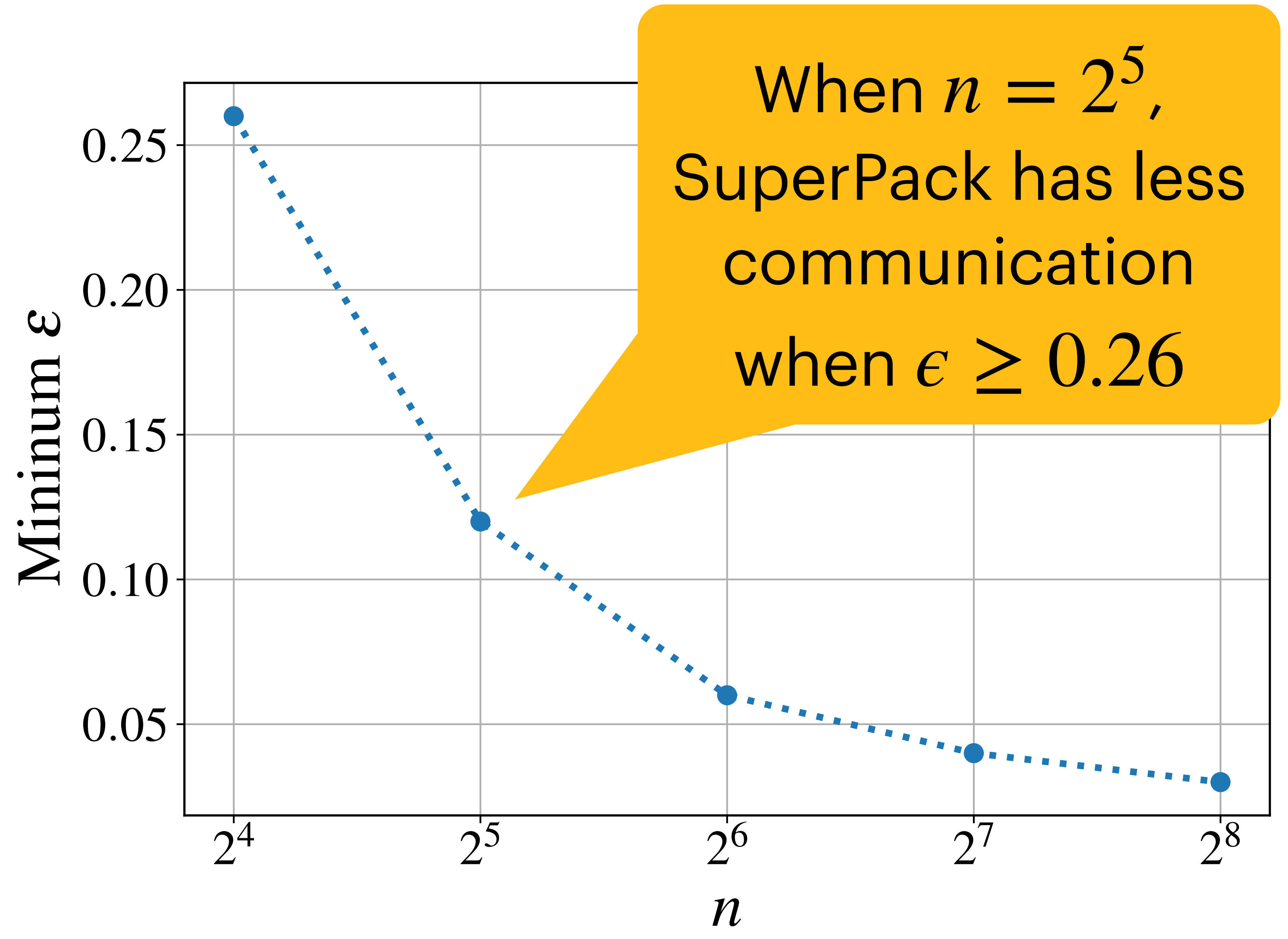
The larger n,
the less honest parties
needed to outperform Turbospeedz.

Turbospeedz: Double your online SPDZ! Improving SPDZ using function dependent preprocessing. Ben-Efraim et al. ACNS 2019.
Le mans: Dynamic and fluid MPC for dishonest majority. Rachuri et al. CRYPTO 2022.

Compare with Turbospeedz - Online

Requirements on parameters (n, ϵ) in order to outperform Turbospeedz*





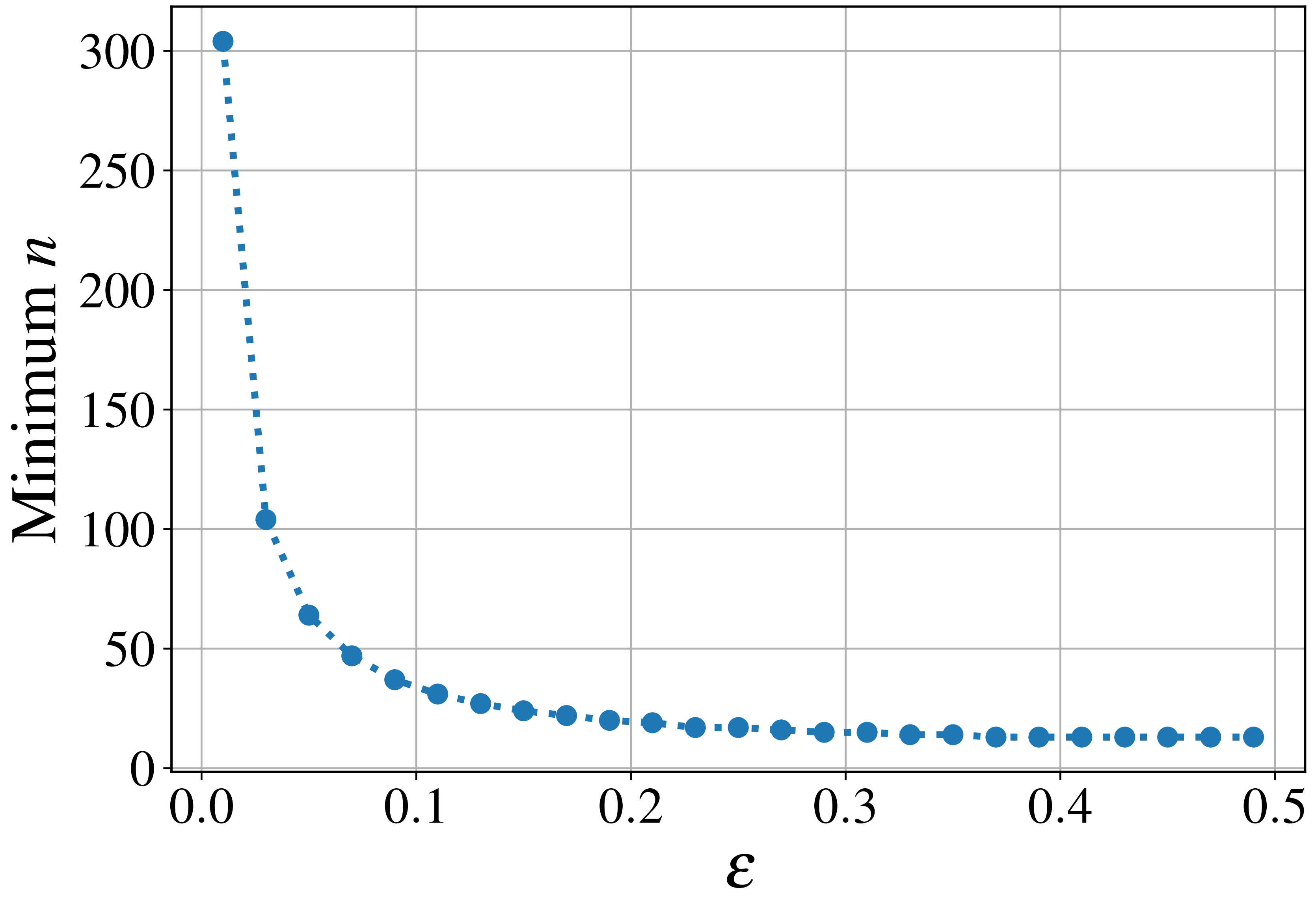
The larger ϵ,
the more honest parties,
the less n needed to outperform Turbospeedz.

The larger n,
the less honest parties
needed to outperform Turbospeedz.

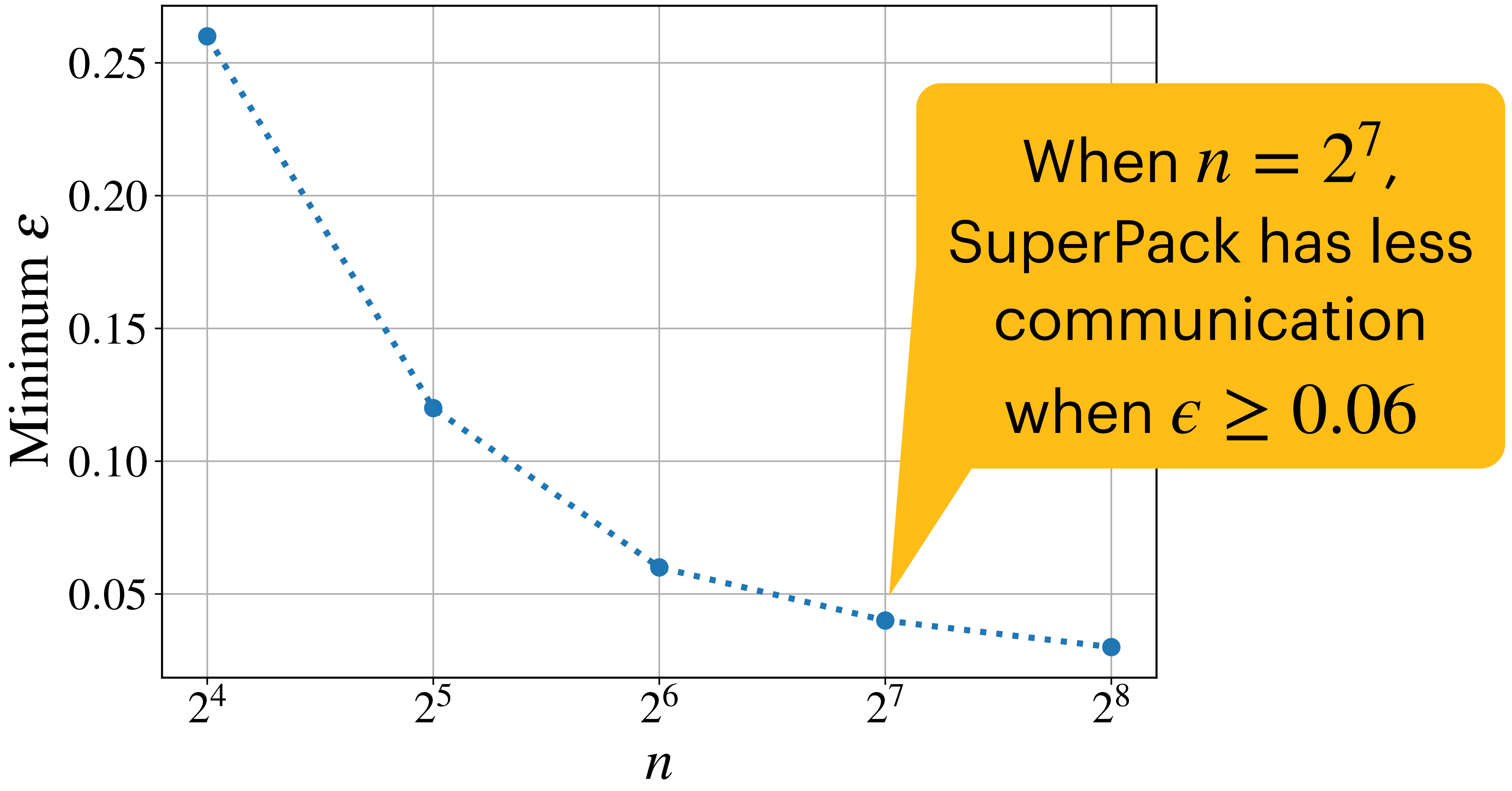
Turbospeedz: Double your online SPDZ! Improving SPDZ using function dependent preprocessing. Ben-Efraim et al. ACNS 2019.
Le mans: Dynamic and fluid MPC for dishonest majority. Rachuri et al. CRYPTO 2022.

Compare with Turbospeedz - Online

Requirements on parameters (n, ϵ) in order to outperform Turbospeedz*



The larger ϵ,
the more honest parties,
the less n needed to outperform Turbospeedz.

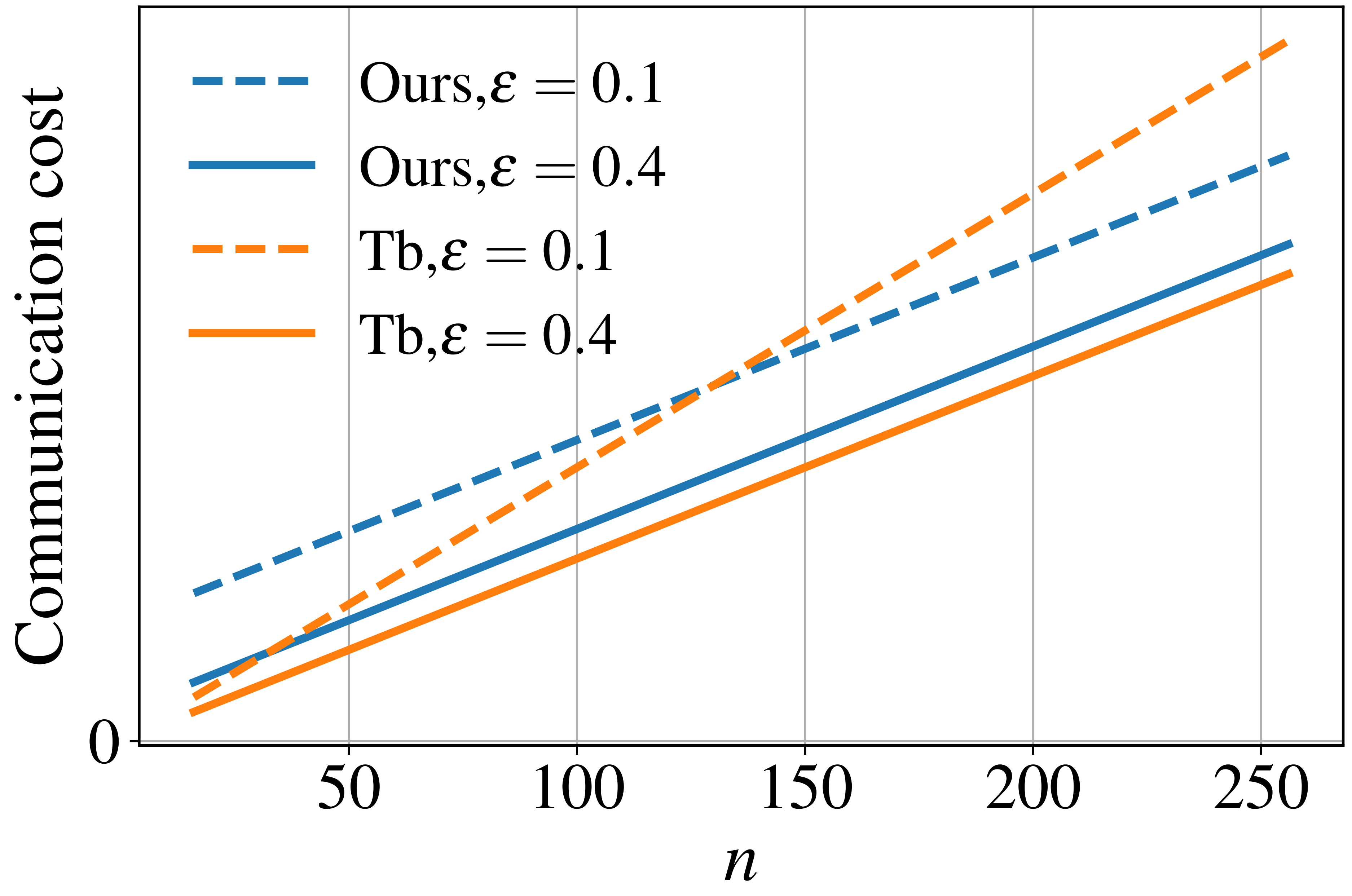


The larger n,
the less honest parties
needed to outperform Turbospeedz.

Turbospeedz: Double your online SPDZ! Improving SPDZ using function dependent preprocessing. Ben-Efraim et al. ACNS 2019.
Le mans: Dynamic and fluid MPC for dishonest majority. Rachuri et al. CRYPTO 2022.

Compare with Turbospeedz - Preprocessing

Communication complexity based on choice of (n, ϵ)



SuperPack has advantage for small ϵ and large n.
The ratio of Turbospeedz / SuperPack is between 0.83 and 1.6.
The cost is reasonable considering the performance gain during online phase.

Turbospeedz: Double your online SPDZ! Improving SPDZ using function dependent preprocessing. Ben-Efraim et al. ACNS 2019.
Le mans: Dynamic and fluid MPC for dishonest majority. Rachuri et al. CRYPTO 2022.

Implementation and Evaluation

Performance evaluation of online protocols - running time factor of $\frac{\text{Turbospeedz}}{\text{SuperPack}}$

Bandwidth	# Parties	Percentage of Honest Parties		
		20%	**30%**	**40%**
500 mbps	**32**	0.68	0.68	0.72
	80	1.27	1.57	1.4
100 mbps	**32**	1.67	1.88	1.95
	80	3.88	4.57	4.56
10 mbps	**32**	2	2.53	2.68
	80	4.56	5.73	6.22
Comm. Factor	**32**	1.71	2.24	2.56
	80	4.27	5.6	6.4

Communication ratio.

Implementation and Evaluation

Performance evaluation of online protocols - running time factor of $\frac{\text{Turbospeedz}}{\text{SuperPack}}$

Under high-bandwidth network, the computation is the bottleneck.

Our implementation can be further improved with e.g. FFT.

Bandwidth	# Parties	Percentage of Honest Parties		
		20%	30%	40%
500 mbps	32	0.68	0.68	0.72
	80	1.27	1.57	1.4
100 mbps	32	1.67	1.88	1.95
	80	3.88	4.57	4.56
10 mbps	32	2	2.53	2.68
	80	4.56	5.73	6.22
Comm. Factor	32	1.71	2.24	2.56
	80	4.27	5.6	6.4

Implementation and Evaluation

Performance evaluation of online protocols - running time factor of $\frac{\text{Turbospeedz}}{\text{SuperPack}}$

Bandwidth	# Parties	Percentage of Honest Parties		
		20%	**30%**	**40%**
500 mbps	**32**	0.68	0.68	0.72
	80	1.27	1.57	1.4
100 mbps	**32**	1.67	1.88	1.95
	80	3.88	4.57	4.56
10 mbps	**32**	2	2.53	2.68
	80	4.56	5.73	6.22
Comm. Factor	**32**	1.71	2.24	2.56
	80	4.27	5.6	6.4

Under low-bandwidth network, the communication is the bottleneck.

The comparison of the performance aligns with analysis.

SuperPack Protocol

Background

- Parameters n, k, d.

Number of parties, packing parameter, degree.

- Packed Shamir secret sharing: $[\mathbf{v}]_d$ for $\mathbf{v} \in \mathbb{F}^k$.
  - $k + n$ distinct values $\alpha_1, \ldots, \alpha_k, \beta_1, \ldots, \beta_n \in \mathbb{F}$.
  - Define a degree-d polynomial satisfying
    - $f(\alpha_i) = v_i, i \in [k]$.
    - For $i \in [n]$, party P_i learns $f(\beta_i)$.

SuperPack Protocol

Background

- Parameters n, k, d.
- Packed Shamir secret sharing: $[\mathbf{v}]_d$ for $\mathbf{v} \in \mathbb{F}^k$.
  - $k + n$ distinct values $\alpha_1, \ldots, \alpha_k, \beta_1, \ldots, \beta_n \in \mathbb{F}$.
  - Define a degree-d polynomial satisfying
    - $f(\alpha_i) = v_i, i \in [k]$.
    - For $i \in [n]$, party P_i learns $f(\beta_i)$.

Evaluation points.

SuperPack Protocol

Background

- Parameters n, k, d.
- Packed Shamir secret sharing: $[\mathbf{v}]_d$ for $\mathbf{v} \in \mathbb{F}^k$.
  - $k + n$ distinct values $\alpha_1, \ldots, \alpha_k, \beta_1, \ldots, \beta_n \in \mathbb{F}$.
  - Define a degree-d polynomial satisfying
    - $f(\alpha_i) = v_i, i \in [k]$.
    - For $i \in [n]$, party P_i learns $f(\beta_i)$.

$\mathbf{v}$ evaluated at $\alpha_1, \ldots, \alpha_k$.

SuperPack Protocol

Background

- Parameters n, k, d.
- Packed Shamir secret sharing: $[\mathbf{v}]_d$ for $\mathbf{v} \in \mathbb{F}^k$.
  - $k + n$ distinct values $\alpha_1, \ldots, \alpha_k, \beta_1, \ldots, \beta_n \in \mathbb{F}$.
  - Define a degree-d polynomial satisfying
    - $f(\alpha_i) = v_i, i \in [k]$.
    - For $i \in [n]$, party P_i learns $f(\beta_i)$.

Each party P_i gets a secret share $f(\beta_i)$.

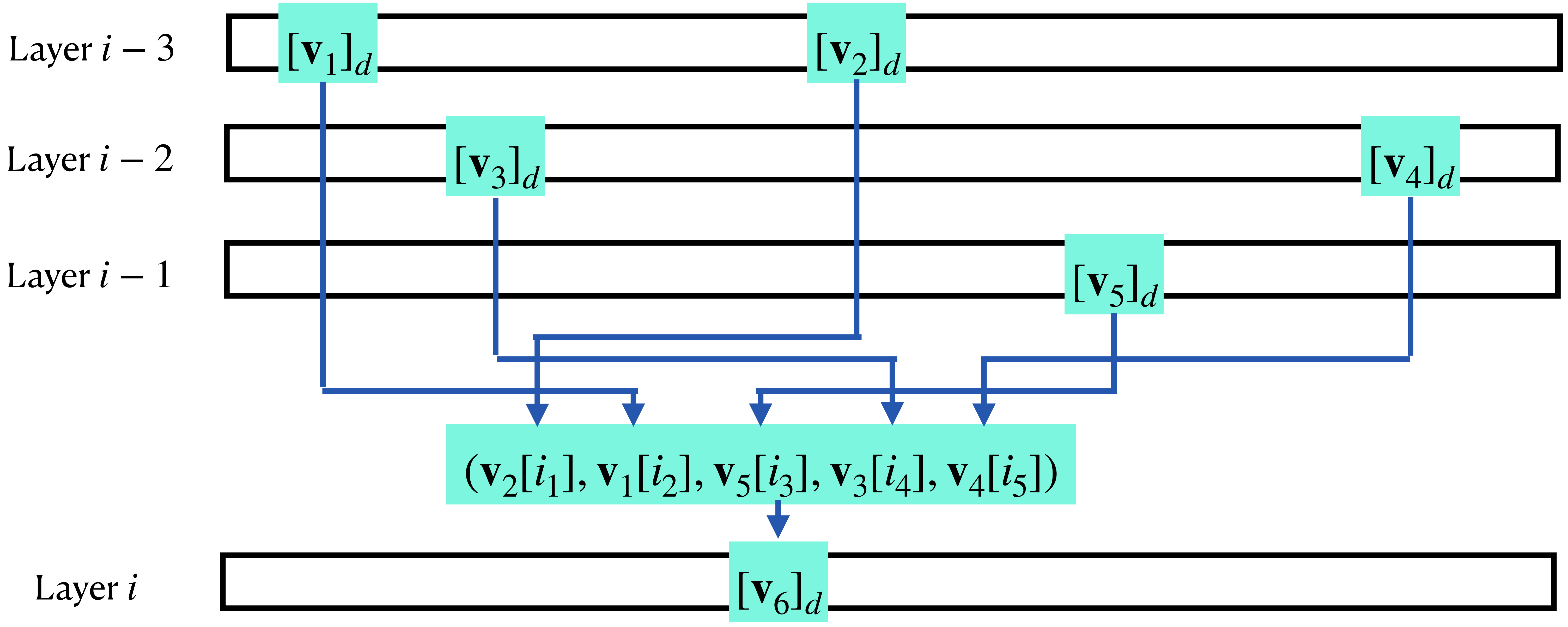
SuperPack: Main Invariant

- For each wire indexed by α with value $v_\alpha \in \mathbb{F}$, sample random $\lambda_\alpha \in \mathbb{F}$
  - λ_α is secret-shared
  - P_1 knows $\mu_\alpha = v_\alpha - \lambda_\alpha$
  - Note: $\mu_\alpha, \lambda_\alpha$ determines v_α

From SIMD to Arbitrary Circuit

- Problem: k wires in each packed share may come from different packed shares from previous layers.

E.g. for batched gate input wires $\overrightarrow{\alpha}$, needs $[\overrightarrow{\mu_\alpha}]_{k-1}, [\overrightarrow{\lambda_\alpha}]_{n-k}$



From SIMD to Arbitrary Circuit

- Problem: k wires in each packed share may come from different packed shares from previous layers.
- Idea: Follow the framework of TurboPack to maintain wiring consistency.

[EGPS22] TurboPack: Honest majority MPC with constant online communication. Escudero et al. ACM CCS 2022.

From SIMD to Arbitrary Circuit

- During the circuit-dependent preprocessing:
  - Prepare shares of $\lambda_\alpha \in \mathbb{F}$ for each wire α
  - Prepare packed shares of $\vec{\lambda}_\alpha \in \mathbb{F}^k$ for each batch of k wires $\overrightarrow{\alpha}$

Communication cost comes from degree reduction.

From SIMD to Arbitrary Circuit

- During the circuit-dependent preprocessing,
  - Prepare shares of $\lambda_\alpha \in \mathbb{F}$ for each wire α.
  - Prepare packed shares of $\vec{\lambda}_\alpha \in \mathbb{F}^k$ for each batch of k wires $\vec{\alpha}$
- During the online phase, for any batch of k wires $\vec{\alpha}$ whose wire values has already been computed:
  - P_1 knows μ_α for any wire α, thus it constructs correct $\vec{\mu}_\alpha$ for this batch

No extra cost for
network routing.

Online Protocol (Simplified)

For each batch of k multiplication gates. Given gate inputs

$$\mathbf{v}_\alpha = \overrightarrow{\mu_\alpha} + \overrightarrow{\lambda_\alpha}, \mathbf{v}_\beta = \overrightarrow{\mu_\beta} + \overrightarrow{\lambda_\beta},$$

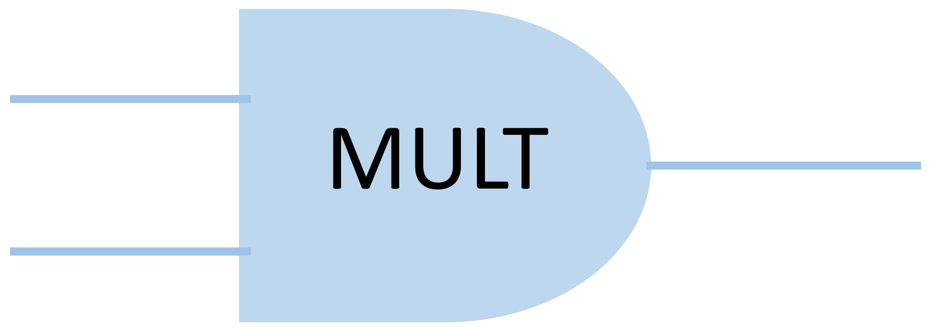
compute gate output

$$\mathbf{v}_\gamma = \overrightarrow{\mu_\gamma} + \overrightarrow{\lambda_\gamma}$$

Online Protocol (Simplified)

$\overrightarrow{\mu_\alpha}, [\overrightarrow{\lambda_\alpha}]_{n-k}$

$\overrightarrow{\mu_\beta}, [\overrightarrow{\lambda_\beta}]_{n-k}$



$\overrightarrow{\mu_\gamma}, [\overrightarrow{\lambda_\gamma}]_{n-1}$

Known from previous gates

P_1 knows

$\overrightarrow{\mu_\alpha}, \overrightarrow{\mu_\beta}$

All parties knows / shares

$[\overrightarrow{\mu_\alpha}]_{k-1}, [\overrightarrow{\mu_\beta}]_{k-1}$

$[\overrightarrow{\lambda_\alpha}]_{n-k}, [\overrightarrow{\lambda_\beta}]_{n-k}, [\overrightarrow{\lambda_\gamma}]_{n-1}$

$[\boldsymbol{\Gamma}_\gamma]_{n-1} = [\overrightarrow{\lambda_\alpha} * \overrightarrow{\lambda_\beta} - \overrightarrow{\lambda_\gamma}]_{n-1}$

Online Protocol (Simplified)

P_1 knows

$\vec{\mu}_\alpha, \vec{\mu}_\beta$

All parties knows / shares

Distributed by P_1

$[\vec{\mu}_\alpha]_{k-1}, [\vec{\mu}_\beta]_{k-1}$

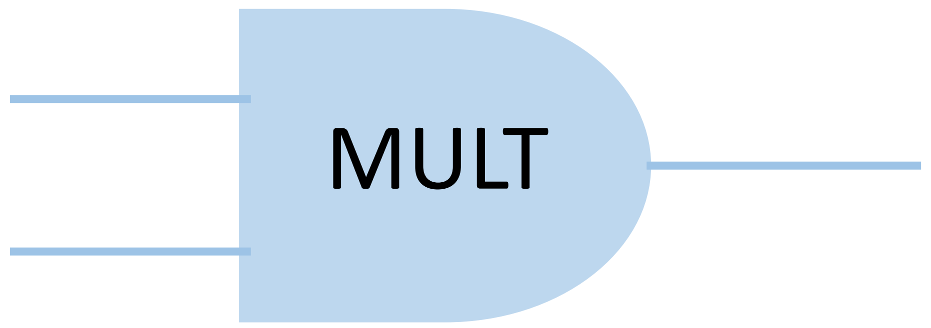
$[\vec{\lambda}_\alpha]_{n-k}, [\vec{\lambda}_\beta]_{n-k}, [\vec{\lambda}_\gamma]_{n-1}$

$[\mathbf{\Gamma}_\gamma]_{n-1} = [\vec{\lambda}_\alpha * \vec{\lambda}_\beta - \vec{\lambda}_\gamma]_{n-1}$

Online Protocol (Simplified)

$\overrightarrow{\mu_\alpha}, [\overrightarrow{\lambda_\alpha}]_{n-k}$

$\overrightarrow{\mu_\beta}, [\overrightarrow{\lambda_\beta}]_{n-k}$



$\overrightarrow{\mu_\gamma}, [\overrightarrow{\lambda_\gamma}]_{n-1}$

P_1 knows

$\overrightarrow{\mu_\alpha}, \overrightarrow{\mu_\beta}$

All parties knows / shares

$[\overrightarrow{\mu_\alpha}]_{k-1}, [\overrightarrow{\mu_\beta}]_{k-1}$

$[\overrightarrow{\lambda_\alpha}]_{n-k}, [\overrightarrow{\lambda_\beta}]_{n-k}, [\overrightarrow{\lambda_\gamma}]_{n-1}$

$[\mathbf{\Gamma}_\gamma]_{n-1} = [\overrightarrow{\lambda_\alpha} * \overrightarrow{\lambda_\beta} - \overrightarrow{\lambda_\gamma}]_{n-1}$

From preprocessing

Online Protocol (Simplified)

Goal: compute $\vec{\mu_\gamma}$ and reveal it to P_1.

- Parties compute

$$[\vec{\mu_\gamma}]_{n-1} = [\vec{\mu_\alpha}]_{k-1} * [\vec{\mu_\beta}]_{k-1} + [\vec{\mu_\alpha}]_{k-1} * [\vec{\lambda_\beta}]_{n-k}$$
$$+ [\vec{\mu_\beta}]_{k-1} * [\vec{\lambda_\alpha}]_{n-k} + [\mathbf{\Gamma}_\gamma]_{n-1}$$

- Parties reveal $\vec{\mu_\gamma}$ to P_1

P_1 knows

$\vec{\mu_\alpha}, \vec{\mu_\beta}$

All parties knows / shares

$[\vec{\mu_\alpha}]_{k-1}, [\vec{\mu_\beta}]_{k-1}$

$[\vec{\lambda_\alpha}]_{n-k}, [\vec{\lambda_\beta}]_{n-k}, [\vec{\lambda_\gamma}]_{n-1}$

$[\mathbf{\Gamma}_\gamma]_{n-1} = [\vec{\lambda_\alpha} * \vec{\lambda_\beta} - \vec{\lambda_\gamma}]_{n-1}$

Online Protocol

P_1 knows

$\overrightarrow{\mu_\alpha}, \overrightarrow{\mu_\beta}$

$\overrightarrow{\lambda_\alpha} * \overrightarrow{\lambda_\beta}$ is computed by a packed Beaver triple during preprocessing.

The actual online phase of SuperPack combines:

1. The computing of $\overrightarrow{\lambda_\alpha} * \overrightarrow{\lambda_\beta}$ via packed Beaver triple
2. The computation of $\overrightarrow{\mu_\gamma}$

   Thus <u>reduces communication overhead</u>

All parties knows / shares

$[\overrightarrow{\mu_\alpha}]_{k-1}, [\overrightarrow{\mu_\beta}]_{k-1}$

$[\overrightarrow{\lambda_\alpha}]_{n-k}, [\overrightarrow{\lambda_\beta}]_{n-k}, [\overrightarrow{\lambda_\gamma}]_{n-1}$

$[\boldsymbol{\Gamma}_\gamma]_{n-1} = [\overrightarrow{\lambda_\alpha} * \overrightarrow{\lambda_\beta} - \overrightarrow{\lambda_\gamma}]_{n-1}$

Achieving Active Security

- Idea: use message authentication codes (MACs).
- Notations.
  - Shamir secret sharing & value shared at α_i: $[v|_i]_d$
  - Additive secret share: $\langle v \rangle$

[BDOZ11] Semi-homomorphic encryption and multiparty computation. Bundling et al. Eurocrypt 2011.
[DPSZ12] Multiparty computation from somewhat homomorphic encryption. Damgård et al. CRYPTO 2012.

Achieving Active Security

With message authentication codes

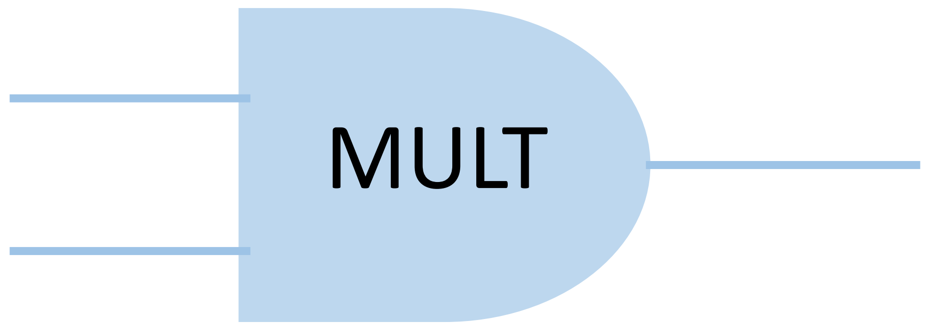
- Secret global key $\Delta \in \mathbb{F}$ shared in the form $([\Delta|_1]_t, \ldots, [\Delta|_k]_t)$.
- Authenticated wire values:

$\mu_\alpha \quad (\langle \Delta \cdot \mu_{\alpha_1} \rangle, \ldots, \langle \Delta \cdot \mu_{\alpha_k} \rangle)$

$[\lambda_\alpha]_{n-k} \quad (\langle \Delta \cdot \lambda_{\alpha_1} \rangle, \ldots, \langle \Delta \cdot \lambda_{\alpha_k} \rangle)$

$\mu_\beta \quad (\langle \Delta \cdot \mu_{\beta_1} \rangle, \ldots, \langle \Delta \cdot \mu_{\beta_k} \rangle)$

$[\lambda_\beta]_{n-k} \quad (\langle \Delta \cdot \lambda_{\beta_1} \rangle, \ldots, \langle \Delta \cdot \lambda_{\beta_k} \rangle)$



$\mu_\gamma \quad (\langle \Delta \cdot \mu_{\gamma_1} \rangle, \ldots, \langle \Delta \cdot \mu_{\gamma_k} \rangle)$

$[\lambda_\gamma]_{n-1} \quad (\langle \Delta \cdot \lambda_{\gamma_1} \rangle, \ldots, \langle \Delta \cdot \lambda_{\gamma_k} \rangle)$

Ways to Obtain Authenticated Shares

- Authenticated additive shares from VOLE.
  - Obtain $\langle v \rangle, \langle \Delta \cdot v \rangle$ via VOLE.
- Random authenticated packed Shamir shares from VOLE.
  - Obtain $\langle \Delta \cdot v \rangle$ via VOLE and locally convert to $[\Delta \cdot \vec{r}]_{n-1}$.
- Authenticated additive shares from authenticated packed Shamir shares.
  - Compute $[\Delta \cdot \mathbf{v}]_d$ and convert to $\langle \Delta \cdot v_1 \rangle, \ldots, \langle \Delta \cdot v_k \rangle$ locally.

Compute Authenticated Value Online (Simplified)

With message authentication codes

Know from previous gates.

P_1 knows

$\overrightarrow{\mu_\alpha}, \overrightarrow{\mu_\beta}$

All parties knows / shares

$[\overrightarrow{\mu_\alpha}]_{k-1}, [\overrightarrow{\mu_\beta}]_{k-1}$

$[\Delta \cdot \overrightarrow{\lambda_\alpha}]_{n-k}, [\Delta \cdot \overrightarrow{\lambda_\beta}]_{n-k}$

$\langle \Delta \cdot (\lambda_{\alpha_i} * \lambda_{\beta_i} - \lambda_{\gamma_i}) \rangle, i \in [k]$

Compute Authenticated Value Online (Simplified)

With message authentication codes

P_1 knows

$\overrightarrow{\mu_\alpha}, \overrightarrow{\mu_\beta}$

All parties knows / shares

Distributed by P_1.

$[\overrightarrow{\mu_\alpha}]_{k-1}, [\overrightarrow{\mu_\beta}]_{k-1}$

$[\Delta \cdot \overrightarrow{\lambda_\alpha}]_{n-k}, [\Delta \cdot \overrightarrow{\lambda_\beta}]_{n-k}$

$\langle \Delta \cdot (\lambda_{\alpha_i} * \lambda_{\beta_i} - \lambda_{\gamma_i}) \rangle, i \in [k]$

Compute Authenticated Value Online (Simplified)

With message authentication codes

P_1 knows

$\overrightarrow{\mu_\alpha}, \overrightarrow{\mu_\beta}$

All parties knows / shares

$[\overrightarrow{\mu_\alpha}]_{k-1}, [\overrightarrow{\mu_\beta}]_{k-1}$

$[\Delta \cdot \overrightarrow{\lambda_\alpha}]_{n-k}, [\Delta \cdot \overrightarrow{\lambda_\beta}]_{n-k}$

$\langle \Delta \cdot (\lambda_{\alpha_i} * \lambda_{\beta_i} - \lambda_{\gamma_i}) \rangle, i \in [k]$

From preprocessing.

Compute Authenticated Value Online (Simplified)

With no extra communication overhead for online phase

- Compute authenticated μ_{γ_i}.

$$\begin{aligned}\langle \Delta \cdot \mu_{\gamma_\mathrm{i}} \rangle = & \langle \Delta \cdot \mu_{\alpha_\mathrm{i}} \cdot \mu_{\beta_\mathrm{i}} \rangle \\ & + \langle \Delta \cdot \mu_{\alpha_\mathrm{i}} \cdot \lambda_{\beta_\mathrm{i}} \rangle \\ & + \langle \Delta \cdot \mu_{\beta_\mathrm{i}} \cdot \lambda_{\alpha_\mathrm{i}} \rangle \\ & + \langle \Delta \cdot (\lambda_{\alpha_\mathrm{i}} \lambda_{\beta_\mathrm{i}} - \lambda_{\gamma_\mathrm{i}}) \rangle\end{aligned}$$

Converted from
$[\Delta \,|_i] * [\overrightarrow{\mu_\alpha}]_{k-1} * [\overrightarrow{\mu_\beta}]_{k-1}$

P_1 knows

$\overrightarrow{\mu_\alpha}, \overrightarrow{\mu_\beta}$

All parties knows / shares

$[\overrightarrow{\mu_\alpha}]_{k-1}, [\overrightarrow{\mu_\beta}]_{k-1}$

$[\Delta \cdot \overrightarrow{\lambda_\alpha}]_{n-k}, [\Delta \cdot \overrightarrow{\lambda_\beta}]_{n-k}$

$\langle \Delta \cdot (\lambda_{\alpha_\mathrm{i}} * \lambda_{\beta_\mathrm{i}} - \lambda_{\gamma_\mathrm{i}}) \rangle, i \in [k]$

Compute Authenticated Value Online (Simplified)

With no extra communication overhead for online phase

- Compute authenticated μ_{γ_i}.

$$\langle \Delta \cdot \mu_{\gamma_i} \rangle = \langle \Delta \cdot \mu_{\alpha_i} \cdot \mu_{\beta_i} \rangle$$
$$+\langle \Delta \cdot \mu_{\alpha_i} \cdot \lambda_{\beta_i} \rangle$$
$$+\langle \Delta \cdot \mu_{\beta_i} \cdot \lambda_{\alpha_i} \rangle$$
$$+\langle \Delta \cdot (\lambda_{\alpha_i}\lambda_{\beta_i} - \lambda_{\gamma_i}) \rangle$$

Converted from
$[\overrightarrow{\mu_\alpha}]_{k-1} * [\Delta \cdot \overrightarrow{\lambda_\beta}]_{n-k}$
$[\overrightarrow{\mu_\beta}]_{k-1} * [\Delta \cdot \overrightarrow{\lambda_\alpha}]_{n-k}$

P_1 knows

$\overrightarrow{\mu_\alpha}, \overrightarrow{\mu_\beta}$

All parties knows / shares

$[\overrightarrow{\mu_\alpha}]_{k-1}, [\overrightarrow{\mu_\beta}]_{k-1}$

$[\Delta \cdot \overrightarrow{\lambda_\alpha}]_{n-k}, [\Delta \cdot \overrightarrow{\lambda_\beta}]_{n-k}$

$\langle \Delta \cdot (\lambda_{\alpha_i} * \lambda_{\beta_i} - \lambda_{\gamma_i}) \rangle, i \in [k]$

Questions

Full version of the paper available at https://eprint.iacr.org/2023/307
Open sourced benchmark available at https://github.com/ckweng/SuperPack